NASA-AISRP ANNUAL PROGRESS REPORT
Report Period: 10/1/2008-9/30/2009
Grant No: NNG05GA30G

Estimating Missing Data in Sensor Network Databases Using Data Mining to Support Space Data Analysis

Submitted to

Dr. Nand Lal
NASA
Phone: (301) 286-7350
Email: Nand.Lal@nasa.gov

By

Le Gruenwald
University of Oklahoma
School of Computer Science
200 Felgar Street, Room 116 EL
Norman, OK 73019
Phone: 405-623-8358
Fax: 405-325-4044
Email: ggruenwald@ou.edu

TABLE OF CONTENTS

1. PROJECT ACCOMPLISHMENTS

In the past year 2008-2009, we were able to complete the following tasks:

- Completed the development of MASTER (Mining Autonomously Spatio-Temporal Environmental Rules), an algorithm to estimate missing sensor data and discover knowledge in centralized sensor networks, taking both the temporal and spatial dimensions of sensor data into consideration.
- Implemented MASTER using C++ and conducted experiments comparing MASTER with four existing estimation algorithms for data streams, WARM, FARM, Spirit and TinyDB, using sensor data gathered from the NASA/JPL Sensor Web project.
- Extended MASTER for multi-hop sensor networks.
- Extended MASTER for cluster sensor networks.
- Developed a data mining algorithm to discover closed frequent itemsets in data streams, SWM (Sliding Window based Mining).
- Investigated additional sensor data applications and gathered additional sensor data including those from the NASA/JPL sensor applications for further testing.
- Graduated two Master's students and prepared two PhD students for their PhD qualifying exams.
- Published two conference papers, submitted two conference papers, and prepared one journal paper for publication submission (twelve publications to date).

In the following sections, we provide the details of the above tasks.

1.1. Completed the development of a framework, called MASTER (Freshness Association Rule Mining), to discover knowledge and estimate missing sensor data in centralized sensor networks, taking both the temporal and spatial dimensions of sensor data into consideration.

MASTER (Mining Autonomously Spatio-Temporal Environmental Rules) is a data mining framework tailored specifically for real-time sensor network applications. The framework allows the online analysis of sensor data streams to uncover the time-and-space dependent dynamic correlations between the data of multiple sensors. This analysis capability is exploited in two ways:

(1) Knowledge discovery
(2) Data estimation

Knowledge Discovery
It is often the case that the intrinsic physical laws governing the environmental variables being monitored by a particular sensor network are unknown and that no prior mathematical idealization for the physical variables exists. MASTER's ability to mine sensor data streams in real-time provides the opportunity to analyze and uncover the secret realities of the environment under watch. The mining performed by MASTER is spatio-temporal; therefore it is capable of capturing the dynamic data changes both in terms of time and space. This capability allows the

discovery of the spatial sensor data dependencies as well as the evolution of such dependencies in terms of time. To discover such knowledge, the user (here researcher or scientist) may compose a mining query involving the sensor information sought. Information contained in a query may be either (1) explicitly stated or (2) generic to some level, in which case MASTER is expected to explore all of the implicit variability assumed in the user query.

Data Estimation

It is a known fact that sensor data are prone to loss, delay, and corruption due to the wireless sensor technology. To compensate for any potential data loss, data estimation is a valuable alternative. The MASTER framework can be sought to serve that purpose. By mining sensor data streams in real-time, spatio-temporal sensor correlations can be constructed dynamically. Hence, at any time if data are to go missing, correlations involving sensors of the missing data can be computed and then the reported data of the correlating sensors can be used to imply data estimates for the missing sensors based on such correlations.

Mining Methodology

The MASTER framework uses and extends the traditional concepts of association rules [Agrawal, 1993] to perform mining in the context of sensor data streams. Traditional association rules were originally developed for discrete data items, also known as basket data. MASTER extends the original association rule constructs to the paradigm of multivariate and continuous data. Succinctly, MASTER assimilates a sensor and a vector subspace into an item. Therefore, an association rule of items refers to set of sensors and their corresponding subspaces that are related. Further, an association rule is qualified as temporal if its data are considered during a well defined temporal period. Formally an association rule is an implication, i.e. a set of sensors and their subspaces implying a different set of sensors and their respective subspaces. An association rule is qualified in terms of two mining parameters: the *support* and the *confidence*. The *support* is the joint probability of the occurrence of all items in the rule. The *confidence* is the joint occurrence of the implied items of the rule (consequent items) conditioned on the antecedent items (those on the opposite side of the consequent items). Both mining parameters must surpass minimum user-defined thresholds.

To accomplish knowledge discovery and/or data estimation, MASTER must compute (mine) all elements of an association rule, i.e. (1) the temporal period, (2) the sets of antecedent and consequent sensors, and (3) the corresponding sensor subspaces.

Data Structure

MASTER owes its mining ability to a data structure, called MASTER-Tree, which sets the stage for the mining algorithms. Although data streams admit an infinite data volume, the MASTER-Tree is a compact structure whose space consumption does not grow in terms of the stream length. That said, the data stored in the tree are actually an abstraction of the raw stream data. The abstract summary is carefully defined to be compact enough without losing key data behaviors. Naturally, the data summary is incremental so as to be additively updated following the arrival of every new data round without increasing the space complexity. MASTER stores the first *m* sample moments of every stream during every fixed period. The MASTER-Tree stores all possible combinations of sensor elements and their corresponding vector subspaces. The complete vector space is assumed to be bounded and it is then discretized in order for the

tree to remain of finite order. Because the power set has an exponential complexity, so does the tree. To prune such complexity, we perform an offline spatial clustering on the complete set of network nodes. The MASTER-Tree is then built separately for each cluster of sensor nodes. The clustering is aware of the associated tree cost and therefore it is made adaptive to user-defined time and space cost bounds. Hence, an arbitrarily chosen compaction and time complexity are assured. On a related note, the estimation algorithm mentioned earlier is made iterative, i.e. the data estimate is fine-tuned progressively. This allows the estimation to be timed out at any time while converging to some estimate, hence also bounding the estimation time. Evidently the more estimation time is allocated, the more the estimate is refined. The iterative character of the estimation procedure is possible by virtue of having an association rule implying one missing node, and then augmenting the rule by adding more relevant nodes to the antecedent rule part.

To handle correlations based on the time domain, we predefine a finite set of elemental cyclic periods. An arbitrary time period is any union composition of two or more elemental periods. A snapshot of the MASTER-Tree collection (one for each cluster) is stored for every period in the predefined period basis. Because data summary is incremental, the MASTER-Tree satisfies an additive property, meaning that if one desires to obtain data during a particular time, it is sufficient to combine (i.e. add together) the needed data from the snapshots that, when composed together, yield the chosen time.

An Iterative Algorithm for Data Estimation

The task of the estimation procedure is to autonomously and efficiently explore the rule space to (1) determine the relevant time period over which data shall be considered for rule evaluation, (2) determine the set of sensor nodes and their respective subspaces that constitute the rule (where the consequent node is the missing node), and (3) compute the estimate of the missing node as its expected value over its consequent subspace. The computed rule is evidently more interesting if its consequent missing node subspace has a "small span" as it would in turn suppress the variance of the estimate. On a related note, the notion of "small" consequent span is arguably a more intuitive way of visualizing the sensitivity of the estimate than the statistical notion of variance (i.e., the average of all squared distances to the mean).

The search over the rule space needs to be properly orchestrated so as our estimation procedure is both effective and efficient. We developed an iterative estimation method in which the estimate is adjusted progressively. The fine-tuning of the estimate can be carried on until the error margin is met or until the estimation execution time is up. This way anytime the control process decides to time out the mining (when the user time bound is reached), we will always have some "ready-to-graduate" estimate. Since a data round may have several missing nodes each having several missing attributes, we shall run different estimation threads each estimating one particular missing attribute of one particular missing node. That way we guarantee that the estimation time is fairly allocated amongst all estimations while each estimation thread progressively fine-tunes its estimate.

Now that we reduced the estimation method to estimating one particular missing attribute only, any estimation-sought rule is only required to imply a singleton consequent node whose subspace shall be viewed and constrained with the respect to the missing attribute in question. Figure 1 shows a flowchart of the estimation method.

Here we provide a summary of the algorithm description, the details of which are given in ([Chok, 2009a], [Chok, 2009b], [Chok, 2009c]). The algorithm starts by identifying the most relevant temporal period for the current estimation problem. The algorithm then obtains the prior distribution of the missing attribute to be estimated from the *MASTER-Tree*. The algorithm then attempts to contain the stretch of such distribution over the user defined error margin while satisfying the *support* and *confidence* thresholds. This rule can be viewed as $\emptyset \rightarrow M$ where M is the missing node (i.e., nothing implies M). If the last step fails to satisfactorily constrain the span of M, then relevant information from other streams needs to be acquired to refine the distribution of M. Meanwhile an estimate can be derived from the rule just obtained, and in that case, the consequent subspace of M has span higher than the allowed minimum span threshold (error bound). In reference to this parameter relaxation, such a rule will be referred to as a relaxed rule. The algorithm chooses one new antecedent node to imply the posterior distribution of M's missing attribute. An initial relevant subspace for the antecedent neighboring node is chosen based on its current reading. If enough *support* cannot be found, the relevant subspace is augmented iteratively until the *support* condition is met. Unless this antecedent node to be added has previously been missing, the subspace augmentation process is guaranteed successful termination. If, however, the relevant subspace for the new antecedent node was increased all the way to reach the limits of the complete subspace V without enough *support* being found, then the potential rule cannot possibly include the current antecedent node, and the algorithm skips it and proceeds to attempt a new antecedent node. After assuring enough rule *support*, the same principle of trying to constrain the posterior distribution of the missing attribute is applied. The new *support* and *confidence* can be efficiently and incrementally recomputed with every change of the *relevant* or *consequent* subspaces.

The integration of a new antecedent neighbor is repeated until the estimation procedure reaches one of the three possible conditions: (1) a rule meeting the minimum *support*, *confidence*, and consequent subspace span is found, (2) the mining is timed-out and a relaxed rule is found, or (3) no more neighbors to add to the antecedent node set and a relaxed rule is obtained. The procedure then returns the estimate value as the final expected value computed over the consequent subspace of the final rule.

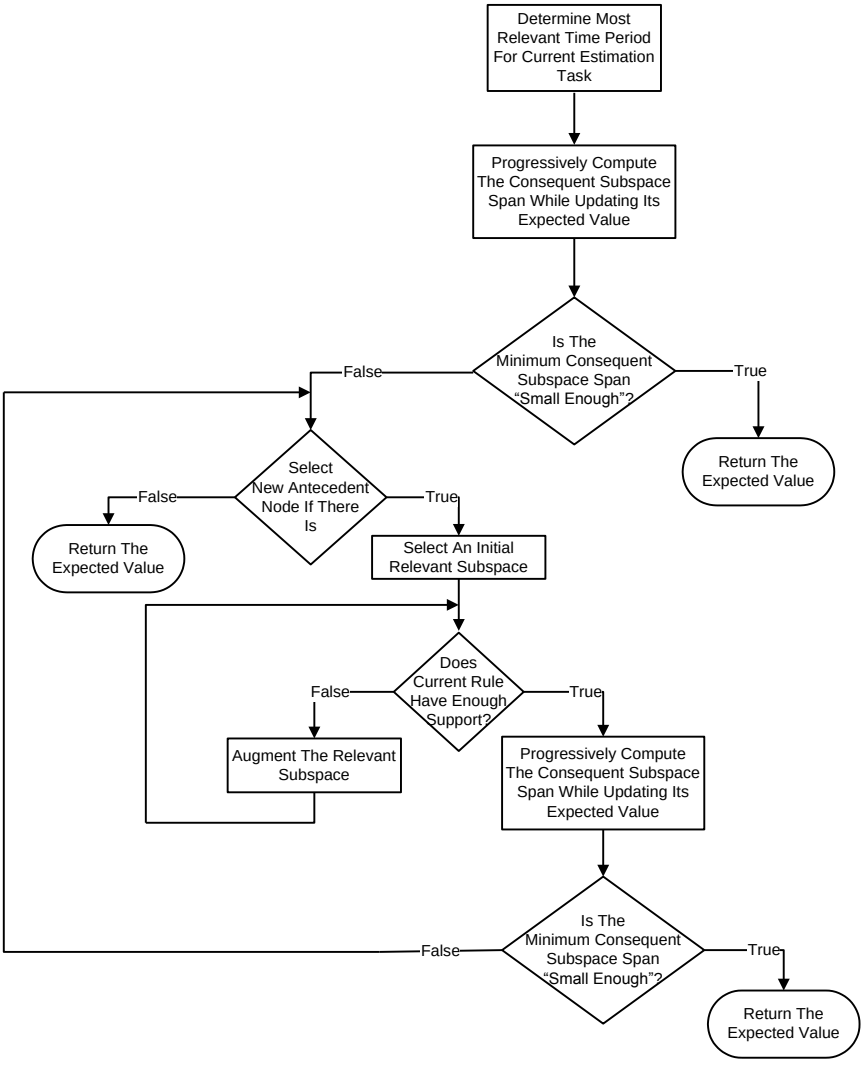

Figure 1. Data Estimation Algorithm

1.2. Implemented MASTER using C++ and conducted experiments comparing MASTER with four existing estimation algorithms for data streams, WARM, FARM, Spirit and TinyDB, using sensor data gathered from the NASA/JPL Sensor Webs project.

We ran simulation experiments to compare our method with the recent works on estimation of missing data streams, namely FARM [Gruenwald, 2007], WARM [Halachev, 2005], SPIRIT [Papadimitriou, 2005] and TinyDB [Madden, 2005]. It was reported in [Gruenwald, 2007] that these techniques perform better than the standard regressive statistical estimation approaches ([Dempster, 1977], [Rubin, 1987], [Rubin, 1996]); hence we do not include those statistical methods in our comparison. To provide a common comparison ground, we let the compared methods view a multidimensional node as virtually separate nodes, each sampling data from one-single attribute stream. To assess the contributions of our temporal mining, and multivariate node

rules, we also ran two additional reduced versions of MASTER where in one we turned off the temporal mining and in another, we only looked at single-dimensional data.

We collected datasets from the real-life NASA deployed sensor network of 12 sensor nodes [NASA, 2009], which monitored the attributes of *temperature*, *humidity*, and *flux* in several locations within the Botanical gardens in San Marino, CA. The streaming rate was once every time tick of 5 minutes. This sensor network is part of NASA's Sensor Webs project. The network's spatial map is depicted in Figure 2. In all experiments, we simulated missing data every *30* (plus a random noise) sensor reading rounds. Also a missing value may be missing for few consecutive rounds. All experiments were run on 2.49 GHz PC with 3.5 GB of RAM memory running a 2002 version of Windows XP Professional.

Figure 2. NASA's Sensor Network Spatial Map [NASA/JPL]

To test the significance and consistency of our results, we collected four datasets, DS1-DS4, during 4 different time periods of variable lengths: DS1 contains samples from a 3-day period, DS2 includes samples from a 1-week period, DS3 a 4-week period, and DS4 a 1-year period. The NASA network sampled data every 5-minutes. We compared all methods on 2 performance metrics: estimation accuracy and average execution time per round. We measured the accuracy

with respect to the Mean Absolute Error (MAE). In addition to the comparative performance experiments, we ran sensitivity and network-size scalability experiments as well as an accuracy time overhead tradeoff experiment.

Table 1 (a). Default Comparative Performances (MASTER Methods Timed Out)

Method\Dataset	DS1			DS2			DS3			DS4		
	MAE	VAR	TIME (ms)	MAE	VAR	TIME (ms)	MAE	VAR	TIME (ms)	MAE	VAR	TIME (ms)
MASTER	0.61	0.22	0.13	0.92	0.51	0.13	1.22	1.33	0.13	2.07	1.68	0.13
MASTER (NT)	1.08	0.91	0.13	1.91	0.88	0.13	2.26	1.81	0.13	4.12	3.12	0.13
MASTER (SD)	0.70	0.45	0.13	1.12	0.54	0.13	1.31	1.35	0.13	2.23	1.72	0.13
FARM	5.48	22.12	12.34	6.06	28.12	13.24	6.44	73.22	11.28	6.89	31.23	18.22
WARM	3.12	1.32	2.65	3.92	1.45	2.97	4.38	2.79	2.66	7.81	3.59	3.51
SPIRIT	9.51	21.06	0.19	10.40	25.66	0.15	12.63	69.53	0.13	18.84	454.69	0.13
TinyDB	81.42	5524.63	0.13	83.86	6038.60	0.13	66.97	4960.49	0.13	58.95	4914.79	0.13

Table 1 (b). Default MASTER Performances (Complete Runtimes)

Method\Dataset	DS1			DS2			DS3			DS4		
	MAE	VAR	TIME (ms)	MAE	VAR	TIME (ms)	MAE	VAR	TIME (ms)	MAE	VAR	TIME (ms)
MASTER	0.24	0.06	0.56	0.38	0.26	0.68	0.66	1.22	0.72	0.92	1.30	1.29
MASTER (NT)	0.55	0.80	0.57	0.69	0.48	0.64	1.07	1.61	0.82	1.45	3.08	1.19
MASTER (SD)	0.45	0.32	0.56	0.54	0.34	0.62	0.62	0.55	0.69	0.89	1.09	1.03

Table 2. MASTER's Performance Sensitivities on DS4 (Complete Runtimes)

Cluster Size	MAE	TIME (ms)	Discretization Granularity	MAE	TIME (ms)	Temporal Granularity	MAE	TIME (ms)
1	2.06	0.16	Very Coarse	1.6	1.89	Coarse	1.13	1.26
2	1.15	0.23	Coarse	1.15	1.57	Fine	0.92	1.29
3	0.98	0.48	Fine	0.92	1.29	Very Fine	0.69	1.31
4	0.92	1.29	Very Fine	0.95	1.21			

Comparative Accuracy Performance

Here we show only the accuracy in estimating the temperature values. Analogous results were observed for other missing attributes. Table 1 (a) shows the MAE of each method over each of the four datasets considered. MASTER's defaults were set to realistic averages (i.e., *minSup*=1%, *minConf*=90%, and 4 nodes per cluster). The defaults of the candidates were chosen either from recommendations in the literature or by extensive experimentation to pick the best parameters. MASTER had consistently the lowest and thus the best MAE. By observing the MAE records in the 4th and 5th rows of Table 1 (i.e., MASTER with non-temporal mining (NT) and MASTER with single dimensional data (SD)), it is evident that the temporal mining contributes more to the accuracy of estimation than does information across attributes. MASTER's default temporal periods consisted of all data snapshots over each 2-hour period (beginning from 12am) of every weather season, hence 12x4 (i.e. 48) snapshots. WARM's and FARM's accuracies trail MASTER's and that of its two derivatives. In all cases their accuracies are orders of magnitudes higher than MASTER's. SPIRIT and TinyDB have much worse errors.

The increase in MASTER's error in terms of the data set is due to the increasing number of records from the 1st to the 4th dataset. However, note that we can, for instance, lower the error over the 4th dataset to about the same error as the 3rd dataset by considering additional temporal snapshots over each month of the year (see Table 2). By this same principle, the error can be lowered as desired by considering more snapshots as appropriate. Generally snapshots definition should be chosen according to the desired query power. Table 1 also shows the variance of the temperature estimate for each method (the VAR columns). The MASTER methods have the

lowest variances. MASTER's variances grow in terms of the dataset size due to the increasing amount of records which causes the posterior data distributions to be more scattered.

Comparative Time Performance
In addition to accuracy, Table 1 (a) shows the average execution time per round. MASTER's runtime was timed out to match that of the fastest method (TinyDB). Again this is possible by virtue of the iterative property. Hence this experiment shows that while our method can be made to run the fastest, it simultaneously achieves the best accuracy which is many orders of magnitude higher than all methods.

Accuracy vs. Time Tradeoff
Table 1 (b) shows the estimation runs of the MASTER methods without timing out i.e., the mining is allowed to complete. Evidently, the error is reduced as more time is allocated. MASTER's MAE is less than one degree Celsius on all datasets while its runtime per data round is in the order of one millisecond, which remains negligible compared to the sampling rate of 5 minutes. It is MASTER's overhead flexibility and superior performance in both the overhead and accuracy that allows it to ensure QoS for sensor network applications.

Performance Sensitivity
Table 2 shows MASTER's sensitivity experiments on the dataset DS4 in terms of the cluster size, discretization granularity, and temporal granularity. The accuracy improves with the cluster size; this is easily explained by the fact larger cluster sizes imply more elaborate rules which also require larger running times. Finer discretization granularity improves both the estimation accuracy and the runtime. This is because higher cell resolution makes posterior distributions less scattered and thus less computational work is needed to refine them. Higher temporal granularities also enhance the estimate while requiring marginally higher time overhead. *Coarse* granularity considered snapshots over every 2-hour period, the *fine* granularity (default) over every 2-hour period of every weather season, and the *very fine* granularity over every 2-hour period of every month.

Time Scalability
Figure 3 shows the scalability of the time performances with the respect to the network size. Holding constant the mining parameters, we ran synthesized versions of the dataset DS4 to have varying number of nodes, i.e. from 12 to 108 in increments of 12. The nearly horizontal derives curves suggest that MASTER, SPIRIT, and TinyDB grow linearly in terms of the problem size with SPIRIT growing at a little higher rate. Our scalability experimentation on WARM and FARM (not depicted due to the major scale difference) confirms the claims in ([Gruenwald, 2007] and [Halachev, 2005]) that they scale quadratically.

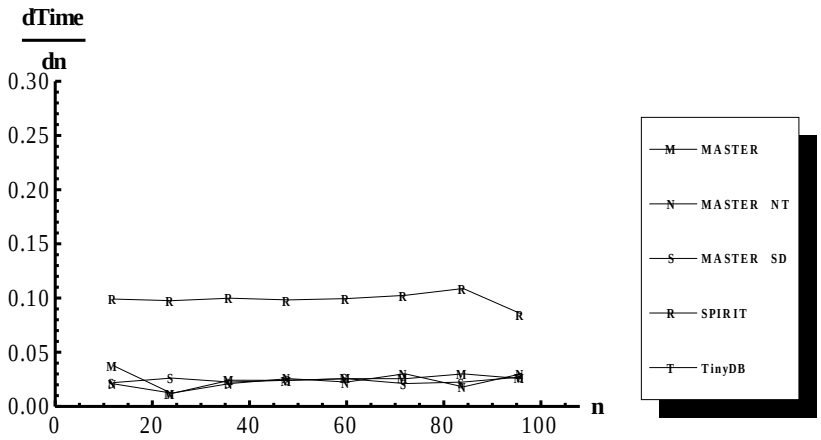

Figure 3. Time Scalability in Terms of Network Size

1.3. Extended MASTER for multi-hop sensor networks

In a multi-hop sensor network, a sensor sends its data to the base station through intermediate sensors. Typically sensors are placed far away from the base station in a multi-hop sensor network. The distant sensors use other sensors to route the data to the base station.

Issues Related to MASTER

In multi-hop sensor networks, many more missing values are generated due to multiple hops in the routing path when compared to single-hop routing, e.g. link failure accumulation, signal interferences, etc. What's more, critical routing node failures may cause all messages routed by these nodes to miss simultaneously. Due to the complexity of such networks, we cannot predict how and when and how many sensors will be missing, and all are dynamic as routing table/paths might change on the fly. These issues drive us to research and develop a new approach for a multi-hop sensor networks environment.

a) Naive Spatial Relation

In single hop centralized sensor networks, spatial relation usually holds due to the homogeneous networks property. Those kinds of networks are typically used to monitor small-scale, stable phenomena. However, multi-hop sensor networks are heterogeneous networks and typically used to monitor complex, large-scale, and changeful phenomena. Figure 4 shows a typical example of heat transferring where a multi-hop sensor network is deployed to monitor some kind of heat transferring process. At a time point, according to the physical phenomenon there are two isotherms that divide the area into three regions. Inside each region, sensors' readings display a similar pattern, i.e. close temperature. However, across regions, sensors' readings vary, what's more, the changing trend of their readings also vary, making it very difficult to uncover the relationships among the sensors from different regions. Apparently if we follow a fixed cluster structure as the one we have done in our MASTER framework for single-hop centralized sensor networks, we risk missing valid association rules among sensors in this example.

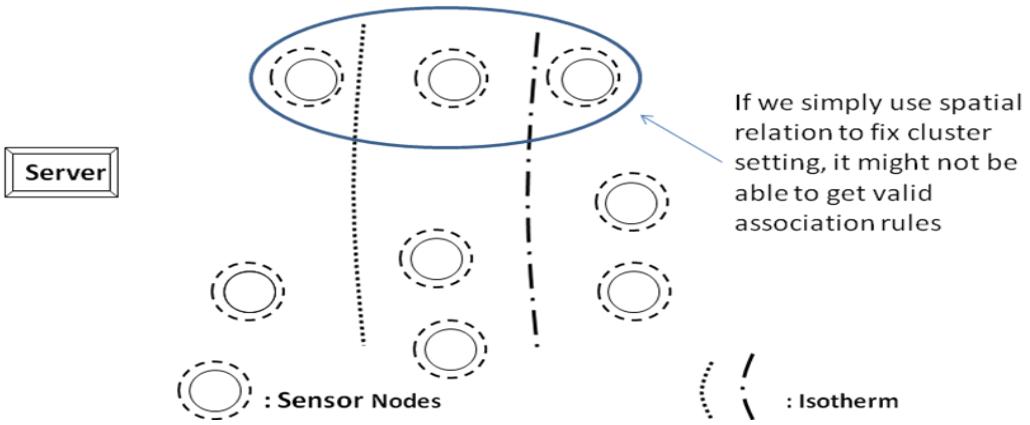

Figure 4. A typical heat transferring process phenomenon

b) Simultaneously Missing

Spatially close sensors are vulnerable to simultaneous failures due to multi-hop routing. The reason is that in a regular multi-hop networks, neighborhood nodes tend to route their messages through a "router" node /gateway /etc., which is often located between its neighborhood and base station. These "router" nodes typically consume more energy due to re-lay messages for other nodes and are more vulnerable to energy exhaustion, message corruptions, etc. What's more, failures of router nodes can cause the loss of whole child nodes' readings/messages, i.e. if we still fix the cluster structure according to spatial positions, then in a cluster all sensors might be missing.

c) Dynamic Routing

In a multi-hop sensor network, routing paths might evolve and networks condition might change on the fly, meaning a fixed cluster structure cannot work well in this situation.

Our Approach

To resolve the issues discussed above, we proposed modifications to the MASTER framework developed for centralized sensor networks that we presented in Section 1.1. Here we summarize the modified framework, called MASTER-M (MASTER for Multi-hop sensor networks), through the following three key modifications:

- Dynamic clustering is necessary for running MASTER on multi-hop sensor networks;
- Initial cluster setting should not use spatial positions as the only criterion; and
- Sensors which tend to be missing simultaneously should be assigned to different clusters.

a) Dynamic Clustering

We first use the training dataset to initialize the cluster structure, and then dynamically adjust the cluster structure on the fly as shown in Figure 5.

- Establish the initial cluster structure through analyzing the training data
- Dynamically adjust the cluster structure on the fly
    - If in a cluster all sensors are missing, adjust the cluster immediately
    - Evaluate sensor relationships in a cluster, if they do not hold any more, adjust the cluster

Figure 5. Dynamic Clustering for multi-hop sensor networks

The Initial Cluster Construction Algorithm

Figure 6 shows the algorithm to construct the initial cluster. At the beginning, arrange all sensors with the most missing first and the least missing last. Here we define: missing rate = # of missing rounds / # of total rounds. Sort the sensors according to the descending order of their missing rate. The sensors with the highest missing probability will be the "seeds" of the clusters. The significance of seeds is threefold: for a clustering technique, choosing the center (seed) of each cluster is typically difficult and important; for data estimation, seed is meaningful for the task, i.e., seeds are the most demanding nodes; and for network routing, seed usually is the sensor node which is far away from the base station.

Next, for each pair of nodes, we compute a distance between the two nodes. There are two types of distances between two sensors: Standard Deviation of Difference of readings (SDOD) and Simultaneously Missing Rate (SMR). SDOD and SMR both are very important for deriving association rules and further estimating missing values. SDOD shows the degree that two sensors are related to each other: the less SDOD is, the stronger the relationship between the two sensors is. SMR shows whether two sensors tend to miss simultaneously: the less SMR, the less chance that two sensors will be missing together.

We normalize SDOD and SMR to a value between 0 and 1, and use the Euclidean distance formula to obtain the distance of SDOD and SMR between two nodes. In this way, we get a digitized measurement of the priority/benefit of putting two nodes into the same cluster. Then we can establish a half matrix of all node pairs to store the distance information. Note that the elements $\{S_i, S_j\}$ and $\{S_j, S_i\}$ are the same due to the symmetry of the distance function; thus we do not need a full matrix.

$\{S_1,S_2\}$	$\{S_1,S_3\}$	. ..	…	$\{S_1,S_n\}$
$\{S_2,S_3\}$	$\{S_2,S_4\}$		$\{S_2,S_n\}$	
…				
$\{S_{n-1},S_n\}$				

- For S_1,(Remember, S_1 has the highest missing rate), among $S_2,\ldots,S_n$, pick up the sensor with the smallest distance, name it as S_i
  - Then for S_2, (if it is already in the cluster of S_1, pick up next node), pick up the sensor with the smallest distance among $S_3,\ldots,S_n$, name it as S_j
    - If $S_j=S_i$, then let S_i decide which distance among S_1S_i and S_2S_i is smaller, use that distance and that pair
    - Then for S_1 (or S_2), choose the second smallest distance
  - Use the above routine, establish 2-node clusters
  - Assume the resource-bounded cluster size is c,
    - For S_1, it has formed a cluster of $\{S_1, S_i,\ldots\}$, if the size of it is less than c
      - For all other nodes, pick up the node with the smallest distance to S_1, for example, S_j
      - Merge $\{S_1, S_i,\ldots\}$ into the cluster containing S_j if the resulting cluster size is not greater than c
    - Do the same thing for $S_2,\ldots,S_n$
    - Repeat the above routine until no new cluster is formed, then the initial cluster structure is done.

Figure 6. The Initial Clustering Construction Algorithm

After the initial structure is initialized, when an essential change occurs, the initial structure is adjusted online as shown in Figure 7.

- Continue to evaluate the relationship between sensors after the cluster structure is initialized
  - Use a FIFO buffer with the size of the training dataset
- When an essential change occurs, adjust the cluster structure
  - Compute the distances on the fly, re-cluster if needed, i.e., if a distance in a cluster is greater than a threshold
- When all sensors are missing in a cluster, adjust the cluster using the re-clustering procedure, which is similar to the initial clustering process.

Figure 7. The Online Cluster Adjustment Algorithm

Empirical Evaluation

For the experimental evaluation of MASTER-M, we employed the real-world dataset - the Intel Lab data [Madden, 2009] - to verify our claim that our approach is better than any known data estimation approach to date. This real life dataset has been collected over a period of approximately one month using an ad-hoc, multi hop routing protocol from a set of 54 sensors (Mica2Dot) placed randomly across a 41 by 31 meter indoor floor space. The hop count as well as the network topology for the associated dataset are variable and based on TinyDB [Madden, 2005] in-network query processing system running on TinyOS operating system. The total number of sensor reading rounds collected for all the sensors is approximately 65,000. However for our experimental evaluation we have used only sensors 41 to 49. This is because these nine sensors have produced the most number of rounds, which is 6,400 rounds in this dataset, which have no missing data. However, we ended up using only the first 3000 rounds (Figure 8) of these

6.400 rounds because as shown in Figure 9, the sensor readings after around round 3000 seem to be unrealistic. We then generated missing data for these rounds, ran MASTER, MASTER-M, TinyDB and Spirit on them, and compared the estimated results with the real data to compute the accuracy of the four methods. Due to un-predictable nature of the underlying routing path that a particular sensor may use to transmit its data, we constructed two arbitrary network graphs, N1 and N2, as shown in Figures 10 and 12, based on a random path and generate missing data from them accordingly. In these figures, each circle represents a sensor node, and B.S. represents the fixed base station where the data estimation algorithms are executed.

Figures 9 and 13 show the performance of the four methods, MASTER, MASTER-M, TinyDB and Spirit, in terms of the error rate in data estimation, corresponding to the two networks N1 and N2. The preliminary results show that overall MASTER and MASTER-M are compatible and perform better than TinyDB and Spirit. One of the reasons that MASTER-M does not show major improvements over MASTER because in this dataset, the expected change in phenomena that is the key motivation for the design of MASTER-M does not exist. Our future plan is therefore seeking datasets that exhibit this characteristic and performing extensive experiments to study its impacts on the behavior of MASTER-M compared with other algorithms.

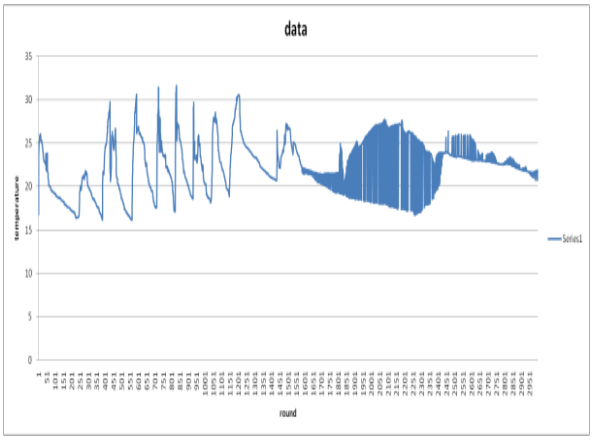

Figure 8. 3000 rounds

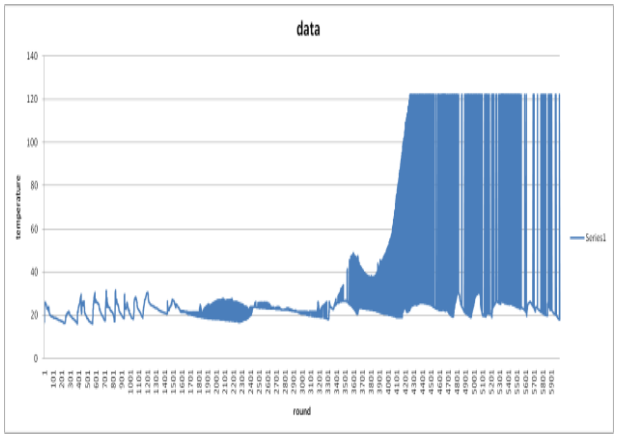

Figure 9. 6400 rounds

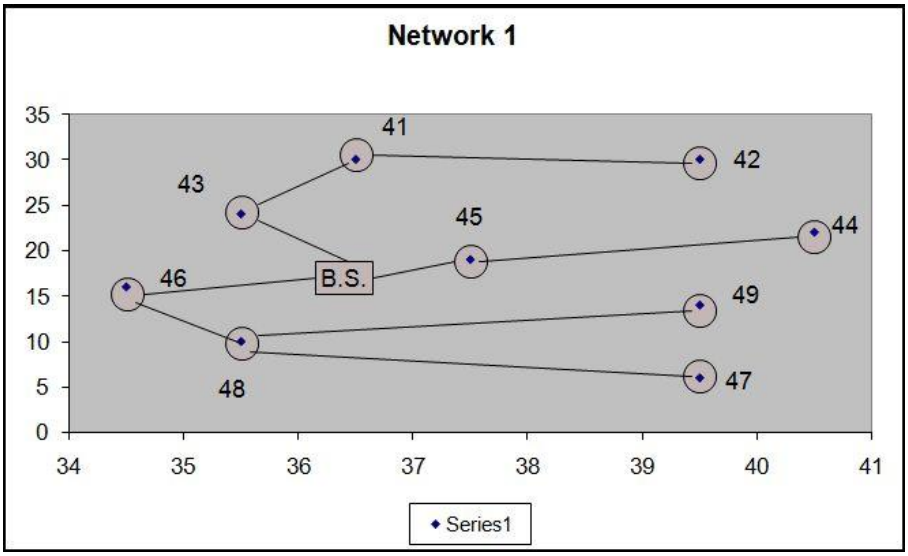

Figure 10. Arbitrary Network N1 and Its Routing Graph

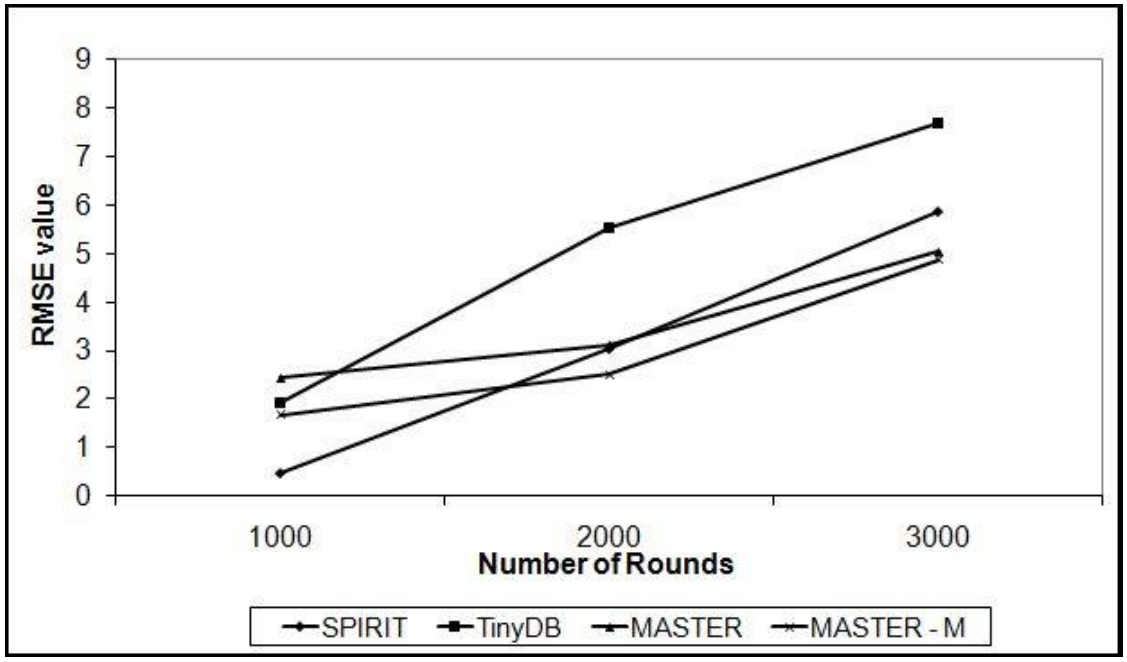

Figure 11. Estimation Error Rates of Data Estimation Techniques in Network N1

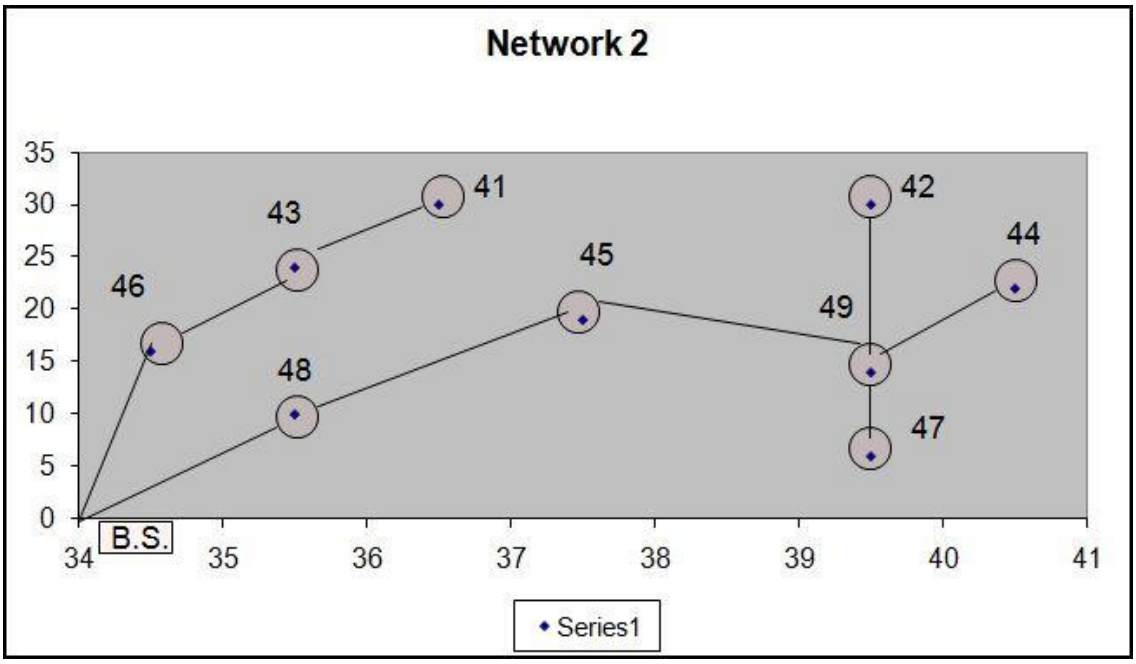

Figure 12. Arbitrary Network N2 and Its Routing Graph

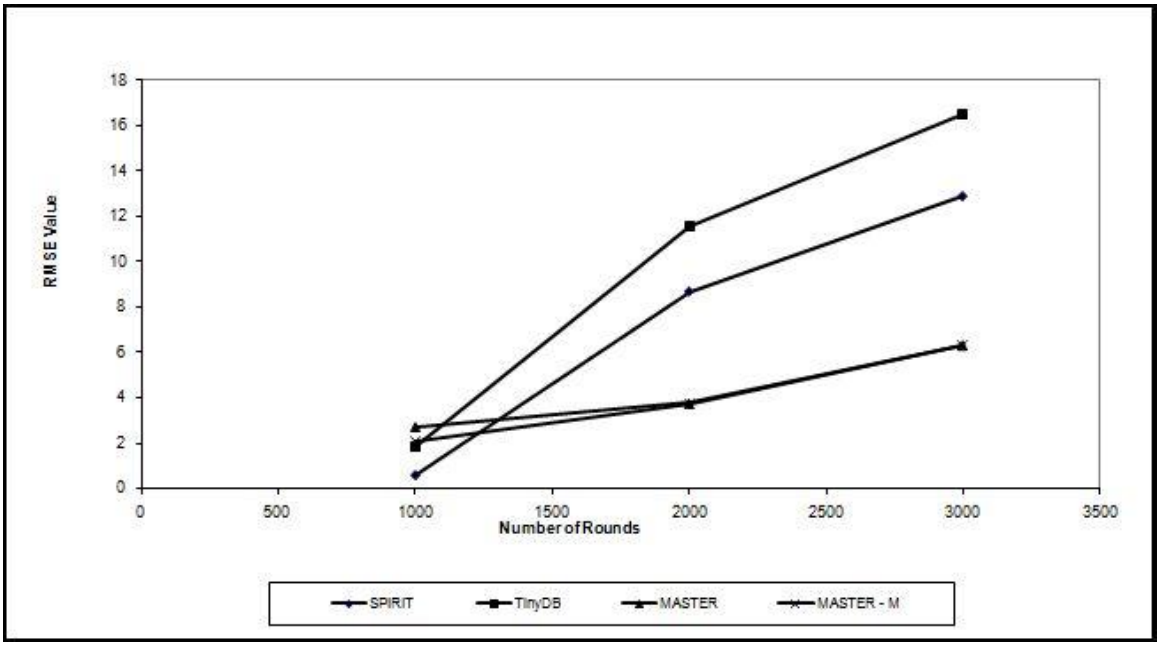

Figure 13. Estimation Error Rates of Data Estimation Techniques using Network N2

1.3. Extended MASTER for Cluster Sensor Networks

Clustered Sensor Network (CSN) is the one where some sensor nodes are clustered into a group to ease data transmission and reduce power usage [Younis, 2004]. Each group has a sensor node designated as a cluster head. In a heterogeneous Clustered Sensor Network the cluster head comes with more resources than the other sensor nodes [Mhatre, 2004] and all the sensor nodes transfer the data through the cluster heads. In some Clustered Sensor Networks cluster heads also work as data aggregation points. A cluster head is the one where data from all the sensors in the group are available; so, we find it justifiable to run the data estimation algorithm at the cluster heads. The major problem with the clustered sensor network is that clusters get changed over time [Mhatre, 2004] because of power failure or equal task distribution or some other reasons. In this section we discuss the weak points of our current data estimation algorithm, MASTER, in Clustered Sensor Network and propose a preliminary solution for it.

Issues Related to MASTER

a) *Change of Cluster Head*
In a clustered network, a node is chosen to be the cluster head that performs as a data fusion point for the cluster [Younis, 2004], [Mhatre, 2004]. No matter homogeneous or heterogeneous clustered networks, cluster heads change over time [Mhatre, 2004]. A clustered head is a lucrative choice to run data mining techniques. This is because all data from other sensors of a cluster is available at this point. Most of the data mining techniques construct a model based on history data and others assume some kind of mathematical model for data distribution. In the former case, if the cluster head changes over time for a sensor, the model needs to transfer to the new cluster head, which requires a transfer protocol. Again if the cluster head fails for some reasons, the model may be destroyed before being transferred to the other node. It is not feasible to replicate a complicated model of online data mining technique in many cluster heads due to the overheads associated with their maintenance.

b) Change of Cluster
Association rule data mining establishes rules among the sensors. If the cluster changes over time [Younis, 2004], two associated sensors in the previous cluster may not be in the same cluster for the newly formed cluster structure. The data mining model must have the capability to partition sensors into small sub-models and later re-combine them again to form a new model. In this case, changing the cluster requires significant work for the cluster heads.

c) Sensor Unavailability
Two correlated sensors may belong to different clusters. They are correlated because they might be close to each other; however, they may belong to different clusters because the corresponding cluster heads are the nearest cluster head from the respective sensors. In such situation, they will go into different clusters even though they are correlated. So if the data mining algorithm runs on a cluster head, then the algorithm will not capture such relationships even though this kind of relationships may be important for data estimation.

Our approach

From the above discussion it is justifiable that incorporating a good re-clustering algorithm into MASTER could be a very good stepping stone toward Clustered Sensor Network. MASTER uses association rule mining with help of a very rigid tree called MASTER tree as described in Section 1.1. MASTER tree is computed at the very beginning of the lifecycle of a sensor network. The tree is very rigid with respect to change. It is very difficult to add a new node or remove a node from the middle of a MASTER tree. In this section, we propose a solution for this problem with necessary modifications to the MASTER tree. Here we assume MASTER runs on the cluster heads and uses the physical clusters as the logical clusters to build the MASTER tree. Our solution algorithm is the solution for building a new MASTER tree from the old available MASTER trees where the new MASTER tree has some sensors from the previous trees. Our intuitive solution tries to exploit the similarities between the sensors with respect to change of data. We call our solution re-clustering because the need of merging the old MASTER trees to form a new one due to physical re-clustering, which essentially leads to logical re-clustering in MASTER.

In our re-clustering approach we assume that each data item has a timestamp attached with it and timestamp is a sequential natural number. We found it justifiable because sensor data are time varying data and they have time attached with them. Another assumption is that the sensor clocks are synchronized with each other. The information available in each node in the current MASTER tree is not enough to establish a new MASTER tree from it because of its lossy nature. However an infinite memory is not acceptable for any lossless data structure designed for data streams; so we add some additional information in each node that will ease the re-clustering process. We add the time interval information within each cell so that we can find the percentage of data items belonging to a particular cell within the attached time interval; later we use that information to reach an intermediate data structure which leads toward re-clustering. Our approach consists of three basis steps:

- Add additional information in each tree node;
- Construct an intermediate data structure to ease the re-clustering process; and
- Develop an algorithm to construct the new MASTER tree.

a) Adding additional information in each tree node

In our modified MASTER approach, called MASTER-C (MASTER for Cluster sensor networks), the MASTER tree has grid-cells in each node and each cell has a range attached with it. Each grid-cell stores the four moments of the data that go to the attached range. We implant two new items in a grid-cell. One is the starting timestamp which stores the timestamp for the first data item that went to that cell, and another one is the ending timestamp which stores the timestamp of the last data item that went to that cell.

b) Constructing a new data structure

We create a novel data structure to carry out our task. Our new data structure is also a tree, which we call a *time interval tree*. A MASTER tree is a descendant of a pattern tree. A MASTER tree uses more than one pattern trees and merges them into one. In that tree a sensor node appears in a couple of places, and the MASTER approach implants a grid-structure on it. We build a time interval tree for each of the appearances of a sensor node. Each of the individual cells in an appearance originates a node in the time interval tree. From now on we will use the term *node* to

represent a tree node of the time interval tree. Each node has four items in it: the starting timestamp, the ending timestamp, the sum of the number of data items the cell has in it, and the children nodes' data items count and a pointer to the originating cell. By using the starting timestamp and ending timestamp we can easily compute how many data items were there within this interval. The amount of data within the interval is actually the number of items an interval node can accommodate.

Definition 1: A node is defined as $c[s,t]$ where *s* is the starting timestamp, *t* is the ending timestamp, and *c* is the items count. The total number of items this node can accommodate is $t-s+1$. A node is called *dense node* if it has as many data items as it can accommodate and a node is called *sparse node* if it has fewer data items than it can accommodate.

Definition 2: An *interval* is a collection of nodes defined by $c(s,t)$ where *s* is the earliest starting timestamp of the nodes, *t* is the latest ending timestamp of the nodes, and *c* is the sum of the items count or the number of data items it has. The total number of items an interval can accommodate is $t-s+1$. An interval is called *dense interval* if it has as many data items as it can accommodate and it is called *sparse interval* if it has few than it can hold. Two or more sparse nodes can form a dense interval.

Definition 3: A node $c[s,t]$ is called a superset node of $c'[s',t']$ if $s \leq s'$ and $t \geq t'$. In that case the latter is the subset node of the former. The same relations held by the superset interval and sunset interval.

Along with the data structures, we want to reveal some of the properties of the nodes and intervals. These properties are used in the algorithm very often. Some of them are too trivial that we have omitted the mathematical construction from this report.

Lemma 1: A dense interval cannot overlap with any other interval.

Lemma 2: A dense interval cannot be in the middle of two overlapping sparse intervals.

Proof: Let $[s_d,t_d]$ is a dense interval and $[s_{s1},t_{s1}]$ and $[s_{s2},t_{s2}]$ are two overlapping sparse intervals. If $[s_{s1},t_{s1}]$ starts before $[s_{s2},t_{s2}]$ and $[s_d,t_d]$ is in the middle of them then $s_{s1} < s_{s2}$ and $s_{s1} < s_d < s_{s2}$. According to Lemma 1, $[s_d,t_d]$ cannot overlap with any other intervals; therefore, $t_{s1} < s_d$ and $t_d < s_{s2}$. Hence, $t_{s1} < s_d < t_d < s_{s2}$ or $t_{s1} < s_{s2}$. If $[s_{s1},t_{s1}]$ and $[s_{s2},t_{s2}]$ are two overlapping intervals then $s_{s2} < t_{s1}$ must be true. But both of the previous arguments cannot be true simultaneously; hence the dense interval cannot be in the middle.

Lemma 3: The dense interval forming sparse intervals must be consecutive.

Proof: This is the more generic case of Lemma 2. We have omitted the proof from this report because of its triviality.

Figure 14 shows the time interval tree construction algorithm. The construction phase of time interval tree starts with creating the root node. The root node has the earliest timestamp from the

MASTER tree as the starting timestamp and the latest timestamp as the ending timestamp. As this node does not represent any cell in the MASTER tree, its data item count is the total number of data items and it does not point to any real cell. At each step the algorithm find the cell with the earliest timestamp from the MASTER tree. Here by using the term MASTER tree we mean the collection of the grids for a sensor's appearance for which the time interval tree is constructed. Then the algorithm creates a new node pointing to that cell. The starting interval, ending interval and data items count of the node come from the cell directly. In the next step, our algorithm finds an appropriate parent for the newly created child node. The parent node is the furthest node from the root node, which is a superset node of the given node. By definition, the root node is the superset node of all nodes. The child nodes under a parent node are sorted according to their starting timestamps. If two or more child nodes form a dense interval, the algorithm will add a new node with the items count as the sum of the items counts of all child nodes and the NULL cell pointers. The starting timestamp (ending timestamp) of the new node will be earliest (latest) among the dense intervals forming the sparse nodes. Those sparse nodes will be the child nodes of the newly formed dense node, and the parent of the newly formed dense node will be the parent of those sparse nodes. Such tree will be built for each of the appearances of the sensors in the MASTER tree.

```
procedure constructTimeTree()
root ← createRootNode()
repeat until all cells are taken care off
  cell ← findNextCellwithEarliestStartingTimeStamp()
  node ← createNewNode(cell)
  parent ← findParent(root, node)
  makeChild(parent, node)
end loop
end procedure

procedure createNewNode(cell) returns node
node ← new node
node.startingTimeStamp ← cell.startingTimeStamp
node.endingTimeStamp ← cell.endingTimeStamp
node.itemCount ← cell.itemCount
node.cell ← c
mark node if densed
end procedure

procedure findParent(root, node)
currentNode ← root
if(currentNode.startTime < node.startTime and
currentNode.endTime > node.endTime)
  foreach childNode in currentNode.children
    if(findParent(childNode, node)!=NULL)
      return childNode
  end loop
else return NULL
end if
end procedure

procedure addChild(parent, node)
parent.children.add(node)
mark node if densed
mergeDensedChildren(parent)
end procedure

procedure mergeDenseChildren(parent)
interval ← empty
nodes ← empty
foreach child in parent.children
  if(! isDense(child))
    if(!interval.disjoint(child))
      interval.join(child)
    else // next nodes are not overlapping with running
interval
      interval ← empty
      nodes ← empty
    end if
    if(isDense(nodes))
      nd ← new node
      foreach node in nodes
        nd.children.add(node)
      end loop
      nd.itemCount = sum of all children item count
      nd.startTime= interval.startTime
      nd.endTime = interval.endTime
      nd.cell = NULL
      mark nd if densed
      interval ← empty
      nodes ← empty
    end if
  else
    interval ← empty
    nodes ← empty
  end if
end loop
end procedure
```

Figure 14. The Time Interval Tree Construction Algorithm

c) Developing an algorithm to construct a new Master-tree

Figure 15 shows our algorithm to construct a new Master tree. The construction process starts with the finding of a minimal size dense node from the trees constructed for the prior sensor node. Once it finds the minimal size dense node, it tries to find a matching dense node for the posterior sensor node. If the algorithm finds an exact match, it builds the MASTER tree for that specific prior and posterior intervals; if it cannot find one, it looks for the smallest superset dense node. The algorithm gets the new node as the smallest node and look for the same size dense node in the prior trees. The algorithm continues until it finds a match. Once it finds a match it collects all the cells, then it removes the cells which are already taken care off. The posterior distribution is computed for individual cells using the Pearson system and those branches are added to the MASTER tree. This algorithm guarantees at least one node because the root node

includes the whole time period and it is the same for all trees; therefore, at least two root nodes will match with each other. Hence, this guarantees the termination of the algorithm.

```
procedure joinMASTER(priorTrees, posteriorTrees)
priorInterval ← empty
posteriorInterval ← empty
intervalUsed ← empty
until all dense intervals are in intervalUsed
  foreach tree in priorTrees
    interval ← findNextSmallestInterval(tree)
    if(interval < priorInterval)
      priorInterval ← interval
    end if
  end loop
  posteriorInterval ← findInterval(posteriorTrees)
  until posterirInterval ≠ priorInterval
    if(posterirInterval > priorInterval)
      priorInterval ← findInterval(priorTrees)
    else
      posterirInterval ← findInterval(posteriorTrees)
    end if
  end loop
  priorCells ← findTheCells(priorInterval) – findTheCells(intervalUsed)
  if(priorCells is a unit cell)
    create posterior grid by finding the cells
  else
    estimate posterior grid
  end if
add the newly created grid with existing grids.
end loop
end procedure

procedure findInterval(trees, interval)
retInterval ← empty
foreach tree in trees
  nodes ← findClosestNodes(tree.root, interval)
  if(nodes – interval < retInterval - interval)
    retInterval ← interval(nodes)
 end if
end loop
return retInterval
end procedure
```

```
procedure findClosestNodes(node, interval)
nodes ← NULL
if(interval subset of node)
  tracking ← nil
  foreach child in node.children
    if(isDense(child))
      if(interval sunset of child)
        nodes ← findClosestNodes(child, node)
        returns nodes
      end if
      if(! Interval.disjoint(child))
        tracking.add(child)
      end if
    else
      if(! Interval.disjoint(child))
        nodes ← allNodes(node)
        returns nodes
      end if
    end if
  end loop
  if(isDense(tracking))
    if(interval subset of interval(tracking))
      nodes ← allNodes(tracking)
    end if
  end if
  if(nodes=NULL)
    nodes ← allNodes(node)
  end if
end if
returns nodes
end procedure
```

Figure 15. The MASTER Tree Construction Algorithm

1.5. Developed a sliding-window based algorithm to mine closed frequent itemsets in data streams.

The underlying data mining approach in our MASTER algorithms is based on the discovery of frequent itemsets in data streams in order to establish association rules among sensors. In this work, we developed an algorithm to discover closed frequent itemsets instead of frequent itemsets. Closure-based mining not only shares attractive features of frequency-based summarization of subsets, but also has outstanding advantages in greatly reducing the number of frequent sets, maintaining complete information and being able to generate non-redundant association rules. Though some approaches directing at the subject have been presented, they basically check all the subsets relevant to the transaction entering or leaving the sliding window and/or the whole complex information mined from the sliding window. This is unnecessary and impractical when transaction size (potential mining space) is large. Moreover, the mutual relationship between additions and deletions of closed itemsets, caused by the continuously entering and leaving transactions, respectively, is ignored in previous works.

Our approach

We developed a novel and succinct strategy for mining closed frequent itemsets over data streams that greatly prunes the checking space to exclude unnecessary candidates. The algorithm is called SWM - Sliding Window based Mining of frequent closed itemsets in data streams. In summary, SWM works as follows. When the window slides, it is accompanied by two transactions: one entering and the other leaving. For either of them, the algorithm first takes it to scan the current window on the fly, intersecting it with all transactions in the sliding window to produce the useful "*closed itemsets*" in order. Then it checks the closed subsets in the data structure (called Closed Trie, CT) to identify these subsets whether they are existing or new closed itemsets. The closed frequent itemsets can be output at any time by traversing CT. The closed itemsets are maintained in a lexicographically ordered trie, where each node is a closed itemset represented by a path from the root to itself. Each node stores two types of information: support and closed support. The algorithm includes the sub-algorithms to add and delete a node from the CT.

Figure 16 shows an example of a sliding window and a current CT. The sliding window has the size W_1 and contains four transactions t_1 (having data items A, B), t_2 (data items C, D), t_3 (data items A, B, C) and t_4 (data items A, B, C). The new transaction t_5 (data items A, C, D) is entering into the sliding window. The pair of two values enclosed in parentheses (x, y) next to a node (e.g. (3, 2) next to node C)) represents the support x and closed support y for that node.

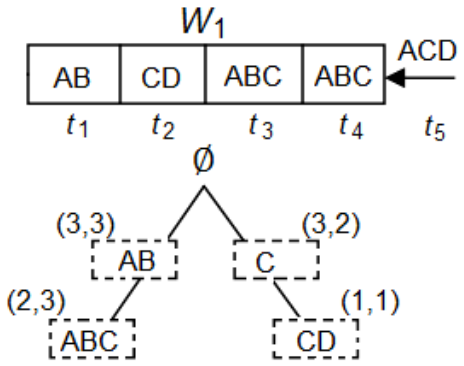

Figure 16. Closed Trie for Current Sliding Window

Performance Evaluation

To validate our algorithm SWM, we first analyzed the existing related algorithms and presented their corresponding computation complexities. Through this analysis and comparison, we demonstrated how SWM improves the efficiency of mining the sliding window-based data streams. Then, we tested our algorithm on both synthetic and real data. The performance of SWM was compared with those of CFI-Stream [Jiang, 2006] and Moment [Chi, 2004], the state of art algorithms for mining frequent closed itemsets in data streams.

a) Computation Complexity Analysis

Moment [Chi, 2004], CFI-Stream [Jiang, 2006] and GC-Tree [Chen, 2008] are online algorithms which perform (frequent) closure checking over data stream sliding windows. To better demonstrate the characters for each of these algorithms, we list the main factors of each algorithm as well as its complexity in Table 4.

In this analysis, it is notable that the time complexity of SWM is much smaller than that of all the previous methods while keeping efficiency in space. The main reason is that it is based on the sliding window size rather than the number of the entire closed itemsets, which is much greater than the former, usually by a magnitude of 10~100 or even bigger.

Table 4. Complexity Analysis

	Main Factor(s) dominating complexity	Complexity	
		Time	Space
Moment	support threshold, sliding window size	$k(t)\cdot n + s\cdot n_c$	n
CFI-Stream	average transaction size, # of closed itemsets	$2^s \cdot k(n)$	n
GC-Tree	# of closed itemsets, sliding window size	$k(t)\cdot n + n_0 \cdot s \cdot l$	$k(m)\cdot n$
SWM	average transaction size, # of new added nodes per transaction	$t\cdot s + n_0^2 + k(k(s))$	n

st-support threshold
s-sliding window size;
t-average transaction size;
n-# of all nodes maintained in the algorithm;
n_0-average # of new added nodes;
m-# of the items;
n_c-# of changes per transaction in Moment
l-average of each item's frequency in the sliding window;
$k(x)$- $k(x)$ depends on x and $k(x) \in [0, x]$, where x is a parameter related to the dataset. Usually $k(x) < x$.

b) Experimental Results

We used six synthetic datasets: T10I4D100K, T15I10D100K, T20I6D100K, T25I8D100K, T30I8D50K and T40I10D10K, denoted by D1~D6, respectively. Each dataset is generated by the same method in [Agrawal, 1994], where the three numbers of each dataset denote the average transaction size (T), the average maximal potential frequent itemset size (I) and the total number of transactions (D), respectively. The performance metrics include the number of checked nodes/itemsets per updating transaction, running time and memory usage. For each dataset, the support threshold for Moment uses the same four values: 0.02%, 0.2%, 0.5% and 1.0% that the authors of Moment used in [Chi, 2004]. For all the experiments we report the average performance over 100 consecutive sliding windows (each with size D).

Sensitivity of Dataset Size Tested with all six datasets D1~D6, the experiment results show that SWM performs the best in terms of the number of checked nodes/itemsets. We only present the results on D4 in Figure 17, but for other datasets, the results are also similar.

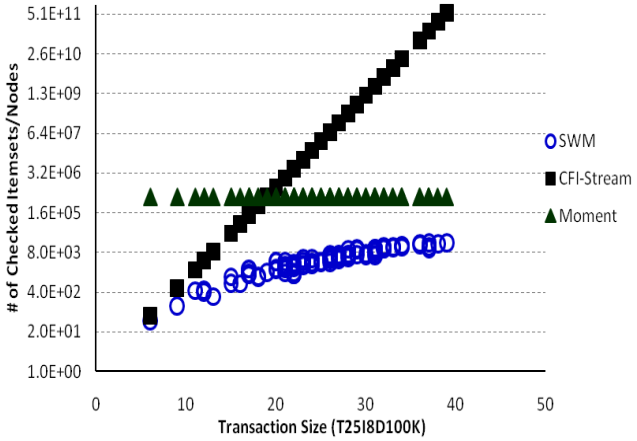

Figure 17. Number of checked nodes per transaction for D4

Performance of Running Time and Space The running time and space on D1~D6 with four different support threshold values are shown in Figure 18 and Figure 19, respectively.

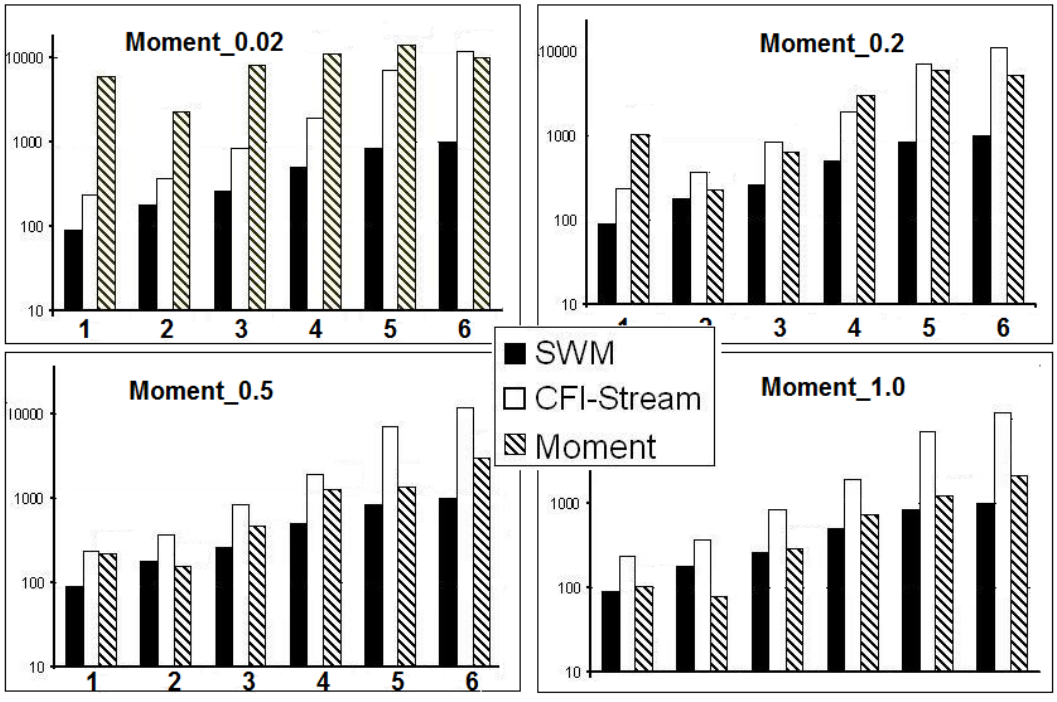

Figure 18. Running Time

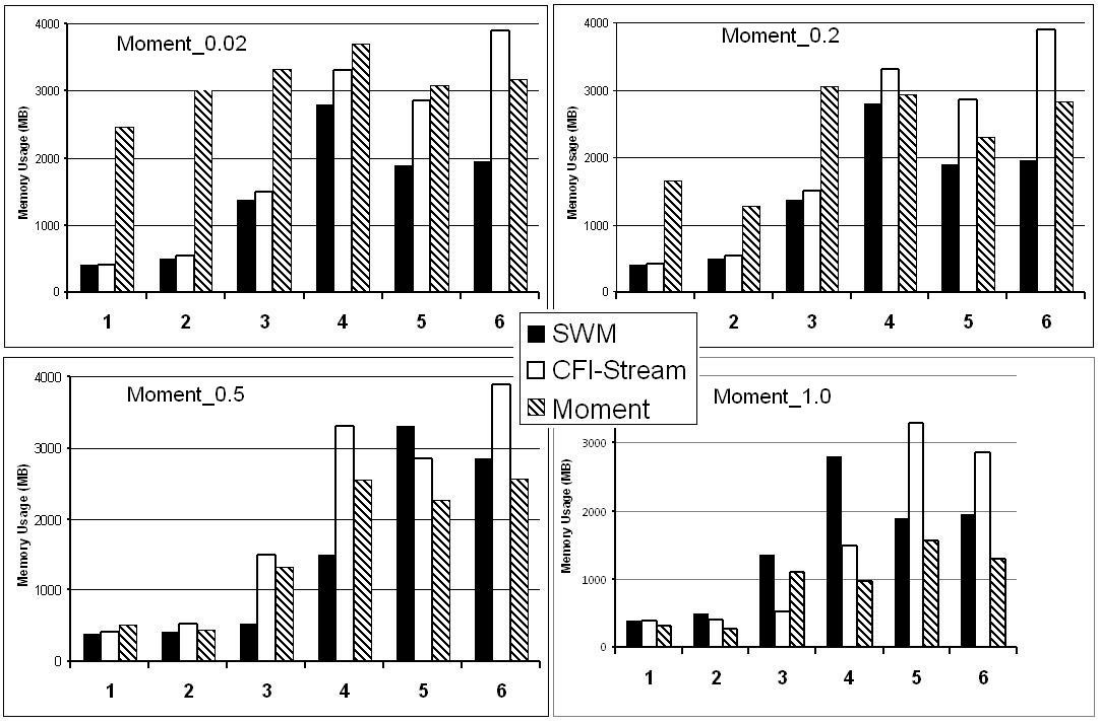

Figure 19. Memory Usage

Both the complexity analysis and experimental results show that our approach is time and space efficient, and has good scalability as the size and number of transactions increase.

2. CHALLENGES

The major challenge that we faced was how to obtain appropriate datasets for our performance studies. We contacted the NASA/JPL research staff and carefully studied their Sensor Webs project in detail. We were able to obtain an appropriate dataset that we could use. We also obtained another NASA/JPL dataset from Dr. Nikunj Oza at NASA Ames Research Center and plan to use it for our next experiments. We are currently also investigating the Sensor Web project from Microsoft which allows many organizations to publish their sensor data and applications on the Web. We hope to be able conduct further experiments with larger real-life datasets to study our algorithms' scalability.

3. PLAN FOR NEXT YEAR

We are requesting a no-cost extension to 7/31/2010 for this project in order to complete it successfully. During the summer months, our student research assistants are allowed to work full-time, and thus, they will be able to produce more research results than during the academic months and help with writing the final report. During the requested extension period from 10/1/2009 to 7/31/2009, we plan to work on the following tasks:

- Extend our data estimation algorithm, MASTER, to mobile sensor networks.
- Conduct theoretical analyses of our MASTER algorithms.
- Incorporate the association rule mining algorithm for data streams that we have developed into our MASTER algorithms.
- Collect real-data sets for mobile sensor network applications.
- Work with NASA scientists to understand the data we obtained and use them for our testing. In particular, we will continue communicating with Dr. Nikunj Oza at NASA Ames Research Center to incorporate the MODIS dataset that he has generously sent to us into our research.
- Conduct theoretical analysis and experiments comparing our MASTER algorithms with existing data estimation techniques for multi-hop sensor networks, cluster sensor networks, and mobile sensor networks using NASA data, other real-life datasets and synthetic data.
- Report the research findings in international conference proceedings and journals.
- Submit the final report to NASA.

4. PUBLICATIONS TO DATE

- Nan Jiang and Le Gruenwald, "Research Issues in Association Rule Mining for Data Streams," SIGMOD RECORD, Vol. 35, No. 1, March 2006.
- Biao Liu, "Classify Data Streams using Concept-Drifting Detection Indicator," Master's Thesis, School of Computer Science, University of Oklahoma, May 2006.
- Nan Jiang and Le Gruenwald, "CFI-Stream: Mining Closed Frequent Itemsets in Data Streams," ACM International Conference on Knowledge and Data Discovery (KDD), August 2006.
- Nan Jiang and Le Gruenwald, "An Efficient Algorithm to Mine Online Data Streams," International Workshop on Temporal Data Mining, August 2006.
- Le Gruenwald, Hamed Chok, and Mazen Aboukhamis, "Using Data Mining to Estimate Missing Sensor Data," IEEE Workshop on Optimization-Based Data Mining Techniques with Applications in conjunction with IEEE International Conference on Data Mining, October 2007.
- Frank Olken, Le Gruenwald, "Data Stream Management: Aggregation, Classification, Modeling and Operator Replacement," IEEE Internet Computing, Vol. 12, No. 6, November/December 2008.
- Hamed Chok, "Spatio-Temporal Association Rule Mining Framework for Estimating Missing Data in Sensor Networks and Analyzing Trend Evolution of Co-Evolving Multidimensional Data Streams," Master's Thesis, School of Computer Science, University of Oklahoma, March 2009.
- Wenyan Wu and Le Gruenwald, "SWM: Sliding Window Based Mining of Frequent Closed Itemsets in Data Streams," submitted to IEEE International Conference on Data Mining (ICDM), July 2009.
- Le Gruenwald, Hamed Chok, Hanqing Yang, and Chandrika Ratnam, "Towards Accurate Estimation of Sensor Data Streams: A Data Mining Approach," submitted to IEEE International Conference on Data Mining (ICDM), July 2009.
- Hamed Chok and Le Gruenwald, "An Online Spatio-Temporal Association Rule Mining Framework for Analyzing and Estimating Sensor Data," accepted for publication in the proceeding of the International Database Engineering and Applications Symposium (IDEAS), September 2009.
- Hamed Chok and Le Gruenwald, "Spatio-Temporal Association Rule Mining Framework for Real-time Sensor Network Applications," accepted for publication in the proceedings of the ACM International Conference on Information and Knowledge Management (CIKM), November 2009.
- Hamed Chok and Le Gruenwald, "Knowledge Discovery and Data Estimation in Sensor Network Databases," under preparation for submission to the IEEE Transactions on Knowledge and Data Engineering, September 2009.

5. REFERENCES

[Agrawal, 1993] Rakesh Agrawal, Tomasz Imielinski, Arun Swami. "Mining Association Rules between Sets of Items in Large Databases", the ACM SIGMOD International Conference on Management of Data, pp. 207-216, May 1993.

[Agrawal, 1994] R. Agrawal, R. Srikant, "Fast Algorithms for Mining Association Rules," Very Large Data Bases, 1994.

[Chen, 2008] J. Chen, B. Zhou, L. Chen, X. Wang and Y. Ding; *Finding Frequent Closed Itemsets in Sliding Window in Linear Time*; IEICE Transactions on Information and Systems, E91-D(10):2406-2418, 2008.

[Chi, 2004] Yun Chi, Haixun Wang, Philip S. Yu, Richard R. " Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window," IEEE Int'l Conf. on Data Mining, November 2004.

[Chok, 2009a] Hamed Chok, "Spatio-Temporal Association Rule Mining Framework for Estimating Missing Data in Sensor Networks and Analyzing Trend Evolution of Co-Evolving Multidimensional Data Streams," Master's Thesis, School of Computer Science, University of Oklahoma, March 2009.

[Chok, 2009b] Hamed Chok and Le Gruenwald, "An Online Spatio-Temporal Association Rule Mining Framework for Analyzing and Estimating Sensor Data," accepted for publication in the proceeding of the International Database Engineering and Applications Symposium (IDEAS), September 2009.

[Chok, 2009c] Hamed Chok and Le Gruenwald, "Spatio-Temporal Association Rule Mining Framework for Real-time Sensor Network Applications," accepted for publication in the proceedings of the ACM International Conference on Information and Knowledge Management (CIKM), November 2009.

[Dempster, 1977] A. Dempster, N. Laird, and D. Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm", Journal of the Royal Statistical Society, Series B, 39 (1), 1977.

[Gruenwald, 2007] Le Gruenwald, Hamed Chok, and Mazen Aboukhamis, "Using Data Mining to Estimate Missing Sensor Data," IEEE Workshop on Optimization-Based Data Mining Techniques with Applications in conjunction with IEEE International Conference on Data Mining, October 2007.

[Halatchev, 2005] M. Halatchev and L. Gruenwald. "Estimating Missing Values in Related Sensor Data Streams," International Conference on Management of Data, January 2005.

[Jiang, 2006] N. Jiang and L. Gruenwald, "CFI-Stream: mining closed frequent itemsets in data streams," ACM International Conference on Knowledge and Data Discovery (KDD), August 2006.

[Madden, 2005] S. Madden, M. Franklin, J. Hellerstein, W. Hong. "TinyDB: An Acquisitional Query Processing System for Sensor Networks." ACM Transactions on Database Systems, 2005.

[Madden, 2009] Samuel Madden, http://db.csail.mit.edu/labdata/labdata.html, Last Access – July, 2009.

[Mhatre, 2004] Vivek Mhatre and Chatherine Rosenberg. "Homogeneous vs. heterogeneous clustered sensor networks: a comparative study," 2004 IEEE International Conference on Communications, 2004.

[NASA, 2009] NASA/JPL Sensor Webs Project, http://caupanga.huntington.org/swim/

[Papadimitriou, 2005] S. Papadimitriou, J. Sun, C. Faloutsos. "Streaming Pattern Discovery in Multiple Time-Series". *Proceedings of the 31st International Conference on VLDB*, 2005.

[Rubin, 1987] D. Rubin, "Multiple Imputations for Nonresponce in Surveys". New York: John Wiley & Sons, 1987.

[Rubin, 1996] D. Rubin. "Multiple Imputations after 18 Years", Journal of the American Statistical Association, 91, pp. 473-478, 1996.

[Younis, 2004] Ossama Younis and Sonia Fahmy. "Distributed Clustering in Ad-hoc Sensor Networks: A Hybrid, Energy-Efficient Approach," INFOCOM 2004, Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies, 2004.